**Практическое задание MPI**

Работу выполнили: Кирьянов С. В., Платонов В. А.

Данная работа, реализованная на ЯП «С++», дополняет работу, реализованную на ЯП «Python», и демонстрирует «Модифицированную каскадную схему суммирования».

**Оглавление**

[**Подготовка рабочей среды** 1](#_Toc185832107)

[**Краткое описание логики работы программы** 4](#_Toc185832108)

[**Результаты работы программы** 5](#_Toc185832109)

[**Код программы** 6](#_Toc185832110)

**Подготовка рабочей среды**

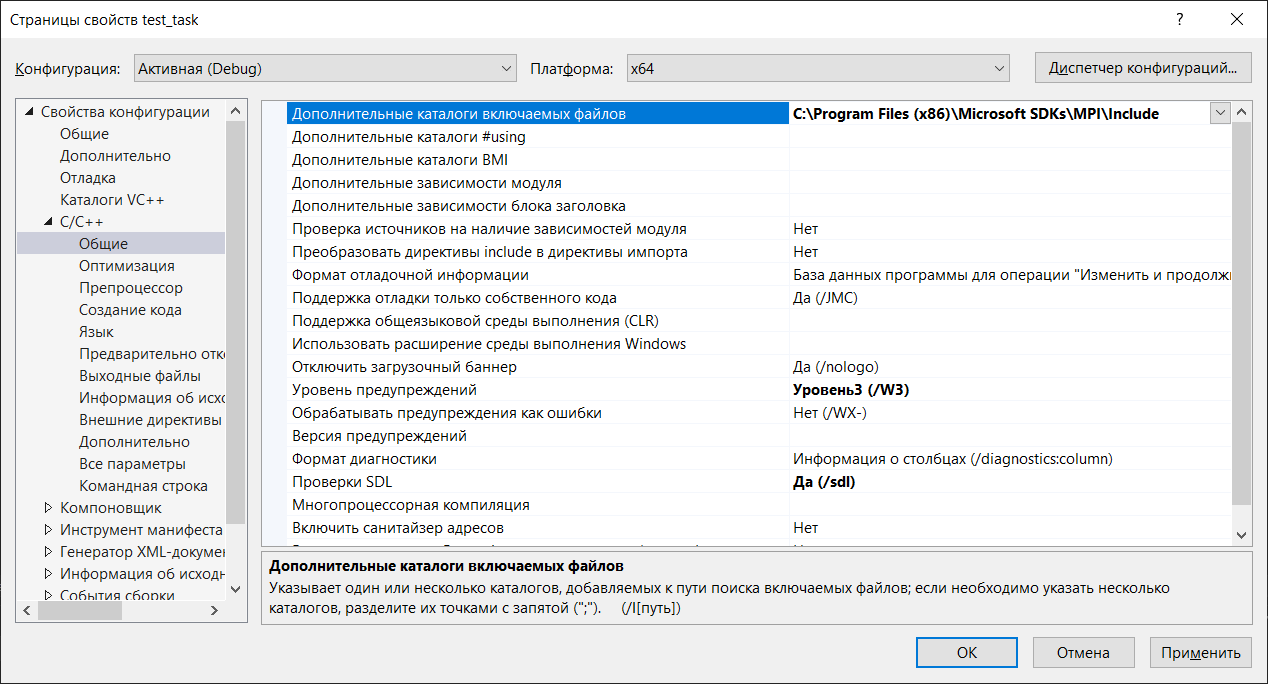
Установить MPI.

Для установки MPI на вычислительный узел необходимо:

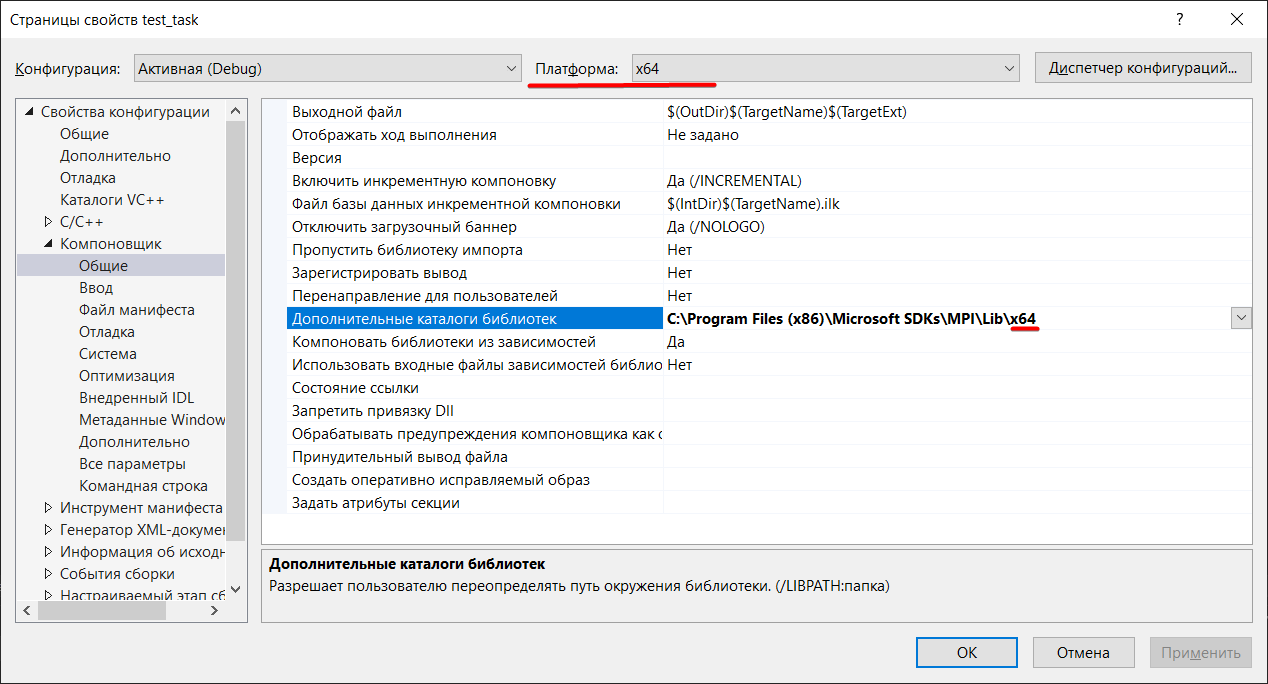
1. Скачать Microsoft MPI с официального сайта - <https://www.microsoft.com/en-us/download/details.aspx?id=105289>.
2. В любом порядке установить скаченные файлы: «msmpisetup.exe», «msmpisdk.msi».

Создать и настроить проект C++:

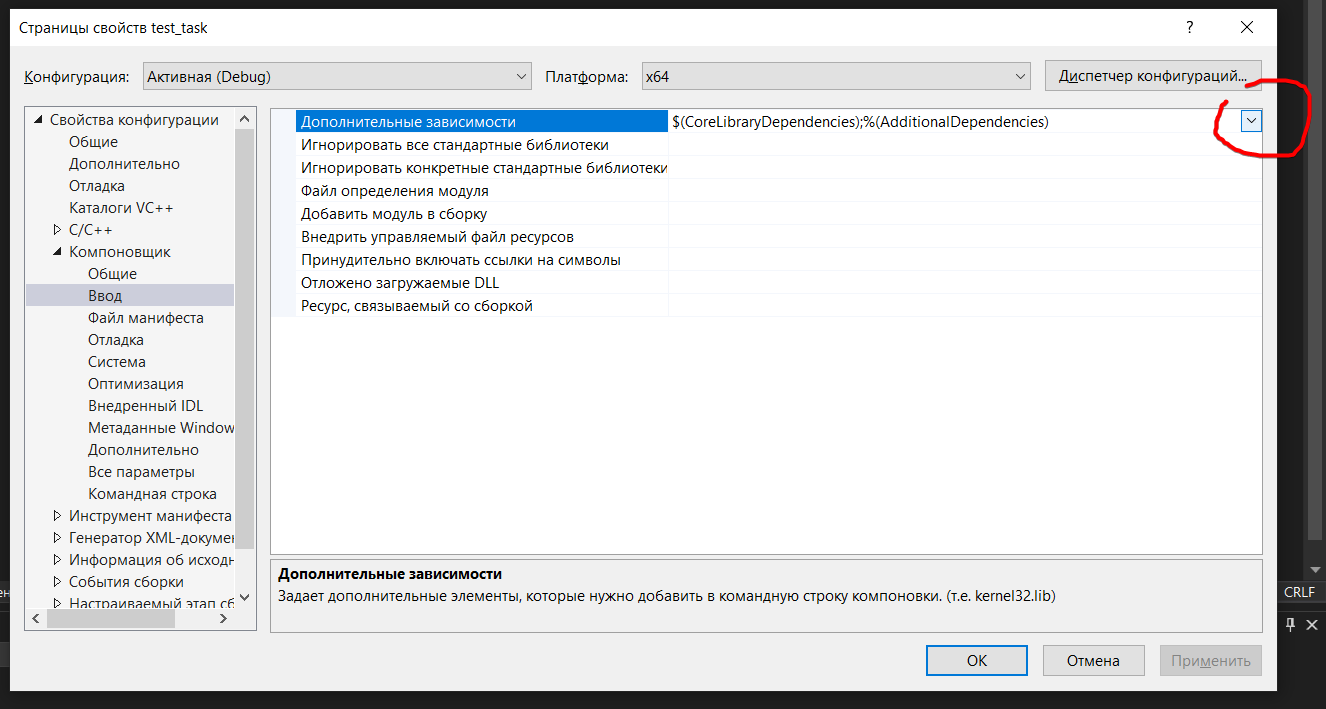
1. Пустой проект;
2. Исходные файлы -> Добавить -> Создать элемент -> Файл C++ (.cpp) (Название файла любое).
3. Зайти в свойства проекта и следовать инструкции, описанной на рисунках 1 – 4.



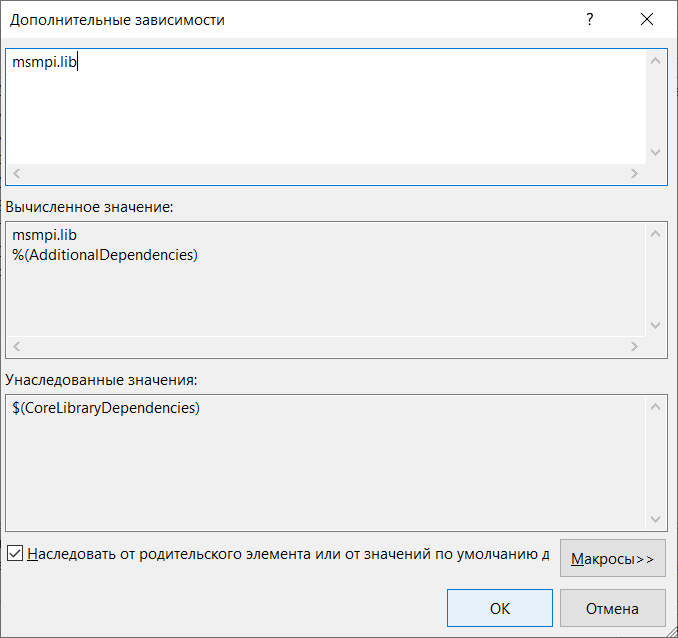
*Рисунок 1 – «Дополнительные каталоги включаемых файлов»*



*Рисунок 2 – «Дополнительные каталоги библиотек»*



*Рисунок 3 – «Дополнительные зависимости»*



*Рисунок 4 – «Ввод дополнительных зависимостей»*

После выполнения всех вышеперечисленных действий, а также заполнения созданного файла с расширением «.cpp» кодом программы, представленным ниже, можно запускать программу.

Для запуска программы программы в многопоточном режиме необходимо открыть консоль и перейти в директорию решения с исполняемым файлов проекта «.exe» (Запуск через «Visual Studio» осуществляется только с использованием одного процесса).

mpiexec -n *КОЛИЧЕСТВО\_ПРОЦЕССОВ* *ИМЯ\_ИСПОЛНЯЕМОГО\_ФАЙЛА\_ПРОЕКТА*.exe

Для запуска программы на разных вычислительных узлах в пределах одной локальной сети необходимо задать дополнительный файл с расширением «.txt» и записать через перевод на новую строку IP адреса или HostName - сетевые имена вычислительных узлов. Все вышеописанные действия необходимо проделать и на подключенных вычислительных узлах:

* #КОПЬЮТЕР\_1 – закомментированный узел (выполнятся не будет).
* NNN.NNN.NNN.NNN.

Затем ввести следующую команду на одном из вычислительных узлов.

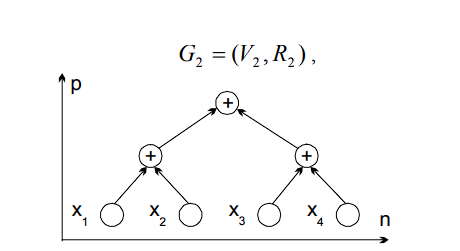
mpiexec -n *КОЛИЧЕСТВО\_ПРОЦЕССОВ* -machinefile *ПУТЬ\_К\_ФАЙЛУ\_С\_АДРЕСАМИ\_УЗЛОВ* *ИМЯ\_ИСПОЛНЯЕМОГО\_ФАЙЛА\_ПРОЕКТА*.exe

**Краткое описание логики работы программы**

Процесс имеющий ранг 0 инициализирует целочисленный массив, состоящий из N количества элементов (N – константа, задаваемая программно). В это время другие процессы попадают на широковещательную функцию «MPI\_Bcast», условием которой является доставка информации до всех процессов, включая наш процесс 0, поэтому остальные процессы ожидают завершения инициализации массива процессом 0 и приступают к выполнению своей инструкции.

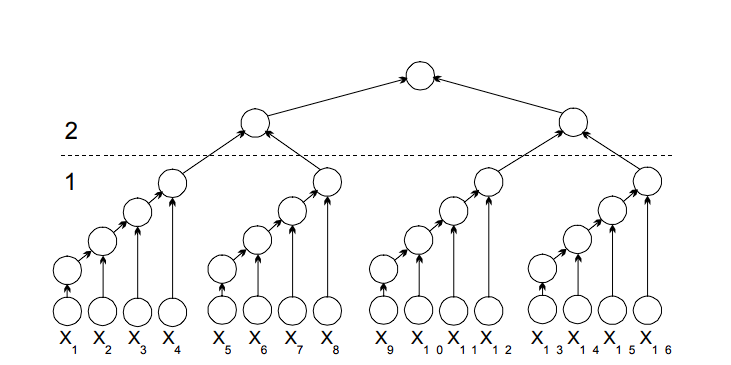
Каждый процесс берет для обработки свой вычисленный интервал массива и суммирует его данные, после чего «MPI\_Reduce» собирает полученные переменные со всех процессов и в процессе сбора при помощи переданного параметра суммируется и результат отправляется главному процессу 0.

При этом при задании M количества процессов равное N количеству элементов массива будет выполнена «Каскадная схема суммирования». Каждому процессу определится по заданному «rank» элемент массива, и данная информацию будет передана для суммирования (рис. 5).



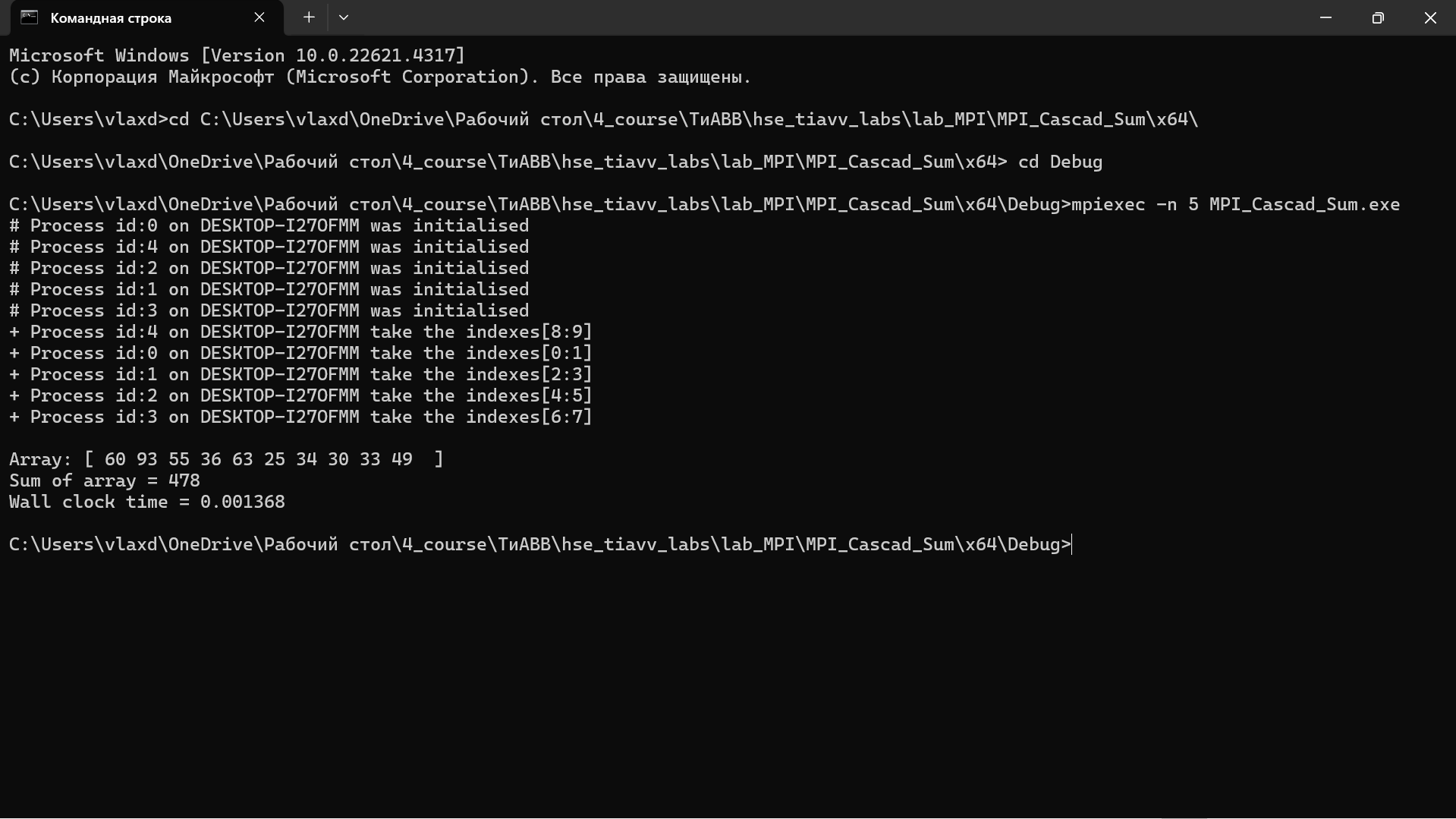
*Рисунок 5 – «Каскадная схема суммирования»*

При задании M количества процессов меньшее N количеству элементов массива будет выполнена «Модифицированная каскадная схема суммирования». Каждому процессу определится интервал, в пределах которого каждый процесс произведет суммирования элементов, и данная информацию будет передана для суммирования (рис. 6).

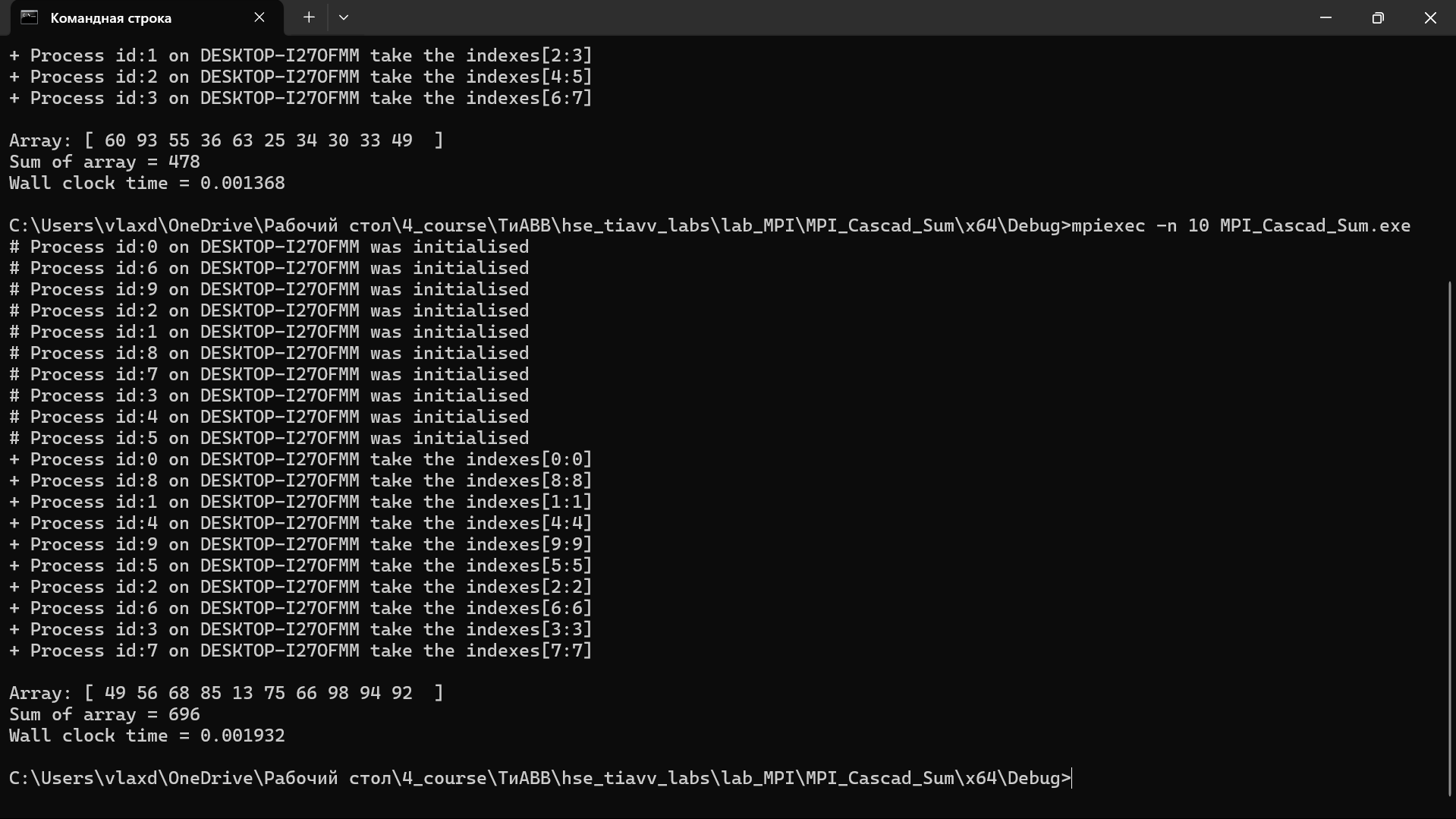


*Рисунок 6 – «Модифицированная каскадная схема суммирования».*

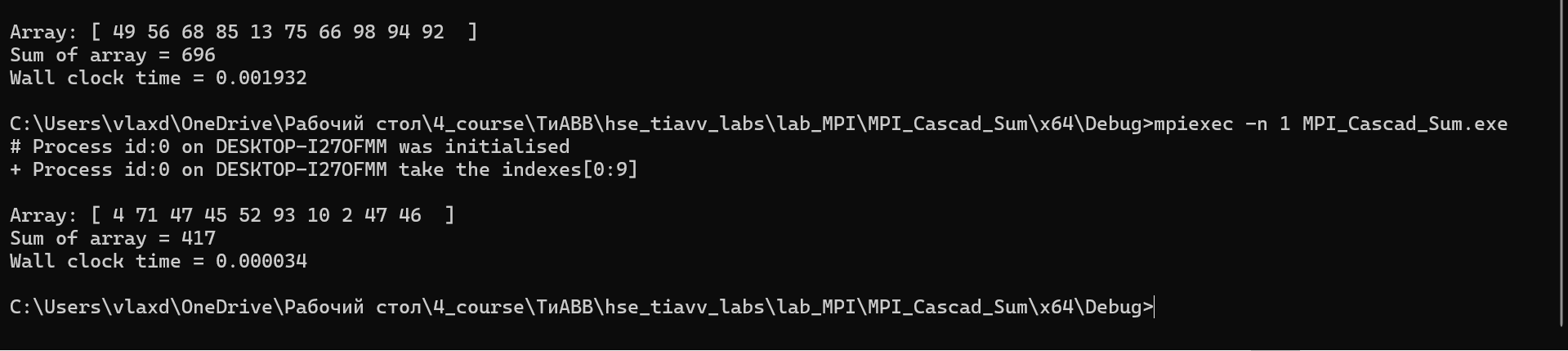
**Результаты работы программы**



*Рисунок 7 – «Массив 10 элементов, 5 процессов»*



*Рисунок 8 – «Массив 10 элементов, 10 процессов»*



*Рисунок 9 – «Массив 10 элементов, 1 процесс»*

**Код программы**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <iostream>

#define N 10

void initArray(int\* arr, int n, int a, int b) {

int i;

for (i = 0; i < n; i++) arr[i] = rand() % (b - a + 1) + a; // генерация случайного числа [a;b];

}

void printArray(int\* arr, int i\_start, int i\_end) {

printf("\nArray: [ ");

for (int i = i\_start; i <= i\_end; i++) printf("%d ", arr[i]);

printf(" ]\n");

}

int main(int argc, char\*\* argv) {

/\*

\* myid - id процессора = rank

\* numprocs - общее кол-во запусщенных процессов

\* i\_start - индекс начала чтения массива для каждого процессора

\* i\_end - индекс конца чтения массива для каждого процессора

\* interval - кол-во элементов для считывания для каждого процессора кроме последенго

(берет все оставшиеся значения)

\*/

int myid, numprocs, i\_start, i\_end, interval;

int arr[N]; // массив с числами для суммирования

/\*sum\_proc

\* sum\_proc - сумма для процессора для его заданного интервала

\* sum\_all - собираемая сумма главным процессором

\*/

int sum\_proc, sum\_all;

double starttwtime = 0.0, endwtime;

int namelen;

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

MPI\_Get\_processor\_name(processor\_name, &namelen);

fprintf(stderr, "# Process id:%d on %s was initialised\n",

myid, processor\_name);

if (myid == 0) {

srand((unsigned)time(NULL)); // меняем сердечник у генератора, берем системное время

int a = 1;

int b = 100;

initArray(arr, N, a, b);

printArray(arr, 0, N - 1);

starttwtime = MPI\_Wtime();

}

MPI\_Bcast(arr, N, MPI\_INT, 0, MPI\_COMM\_WORLD);

interval = N / numprocs;

i\_start = myid \* interval <= N - 1 ? myid \* interval : -1;

i\_end = myid != numprocs - 1 ? i\_start + interval - 1: N - 1;

sum\_proc = 0;

if (i\_start > -1) {

fprintf(stderr, "+ Process id:%d on %s take the indexes[%d:%d]\n",

myid, processor\_name, i\_start, i\_end);

for (int i = i\_start; i <= i\_end; i++) {

sum\_proc += arr[i];

}

}

MPI\_Reduce(&sum\_proc, &sum\_all, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (myid == 0) {

printf("Sum of array = %d\n",

sum\_all);

endwtime = MPI\_Wtime();

printf("Wall clock time = %f\n",

endwtime - starttwtime);

}

MPI\_Finalize();

return 0;

}